数据建模与分析基础

Homework 1 搜索问题

2024 春季学期

2024.1.12

1. 实验说明

本实验由书面部分（回答问题）和编程部分（代码填空）组成。其中：

- **编程部分**：完整代码可在 `code` 文件夹中找到。你只需要在 submission.py 的

```
# BEGIN_YOUR_CODE

# END_YOUR_CODE
```

部分完成代码编写。不要对除 `submission.py` 以外的文件进行更改。建议你在每完成一部分代码填空后按附录的提示对单个样例进行测试，以及时排查代码的问题（例如 `python grader.py 3a-0-basic` ）。在完成所有代码编写后，你可以运行 `python grader.py` 得到最终分数。

> grader.py 中的样例由可见样例和隐藏样例组成，但由于助教手中没有隐藏样例，所以你只需将可见样例全部跑通，且隐藏样例不超时，即可得到相应分数。

- **书面部分**：文件夹中有一个 tex 压缩包（`report.zip`），里面包含了一个叫 main.tex 的文件，建议你使用 Overleaf 创建相应项目过后，根据 main.tex 中标红的 TODO 部分完成书面部分填写。（注：为方便助教批改，编程填空部分的代码也需要复制粘贴到对应位置）

> 本次实验报告需要用 latex 编写，如果你没有使用过 latex，强烈建议你利用网络资源进行相应学习。例如如果你不知道怎么用 latex 语法写公式，你可以在 bing 搜索相关关键词等

在完成所有题目后，你需要在 bb 系统发送一个命名为 学号 _ 姓名 `_HW1.zip` 的压缩包，里面只用包含 submission.py 和 report.pdf 两个文件。助教会据此对你的作业进行评分。**注意事项：**

- **本次作业需独立完成，不允许任何形式的抄袭。如被发现，互相抄了一份作业的几名同学分配此作业的分数。**
- **本次作业的截至时间为** 2024.3.31 23:59:59. **在此之后延迟** 1/2/3/4 **天会扣** 10/30/60/100 **分。**

如果在做作业时任何问题，可以通过 腾讯文档 向助教提问，也欢迎同学帮助解答问题。

2. 实验正文 [90%]

在路线规划中，我们的目标是找到从 A 点到 B 点的最佳方式（例如高德地图）。在这个作业中，我们将建立在经典的最短路径问题之上，以允许更强大的查询。例如，您不仅可以明确询问从第四教学楼到一教金矿的最短路径，还可以询问从科技实验楼返回科大幼儿园的最短路径，途经本科生院、科大附中、一路上还有科大附小（以任何顺序!）。

假设我们有一张由一组位置（location）组成的校园（如南七技校）地图。每个位置都有：

1. 一个独特的标签（例如，6608996258），
2. 指定位置的（纬度、经度）对（例如，37.4299866、-122.175519），以及
3. 一组描述位置类型的标签（tags）（例如，amenity=food）。

位置对之间有一组连接（connection）；每个 connection 都有一个距离（distance）（以米为单位），并且是双向的（如果从 A 到 B 的距离是 100 米，那么从 B 到 A 的距离也是 100 米）。您将使用两个城市地图：网格地图（`createGridMap`）和斯坦福地图（`createStanfordMap`），后者源自Open Street Maps。详情请看`README.md`。我们也提供了 USTC 地图，感兴趣的同学可以尝试（optional）

2.0 统一成本搜索（UCS）、Dijkstra 算法和 A* 算法（18 分）

UCS 和 Dijkstra 算法较为类似，Dijkstra 的算法可能更为人所知，它可以被视为统一成本搜索（UCS）的一种变体，在这种算法中，不存在目标状态，处理过程将继续进行，直到所有节点 (而不仅仅是目标节点) 都从优先队列中移除，也就是说，直到所有节点 (而不仅仅是目标节点) 的最短路径已经确定。

下面将介绍 Dijkstra 算法和 A* 算法的一些概念和问题，方便你更好地理解课上学过的知识。

2.0.1 最短路径问题

引言 搜索问题是经典人工智能的基础问题之一。在本题中，我们将重点关注搜索问题中的最短路径问题，并从算子的角度理解 Dijkstra 算法的正确性（实际上，在后续的强化学习中使用 Bellman 算子也能更好理解价值迭代的过程）。在这里，我们默认正权图 $G(V,E)$ 是有向图，w_{ij} 是边 $(i,j)\in E$ 的权值，我们希望求解源点 s 到任意汇点 t 的单源最短路径 $d_s(t)$。若记序列 $v_0,...,v_{n-1}$ 是顶点集合 V 的一个排列（一定存在，但是未知），并且满足 $d_s(v_k)\le d_s(v_{k+1})$。

一些约定：如果 $(i,j)\notin E$，则 $w_{ij}=+\infty$；$d_s(s)=0$。

引理

$$d_s(u)=min_{v\in V}\{d_s(v)+w_{vu}\}$$

定义 定义 Dijkstra 算子 $\mathcal{D}_k$ 满足

$$\mathcal{D}_k\circ d_s(u)=min_{v\in\{v_0,...,v_{k-1}\}}\{d_s(v)+w_{vu}\}$$

(a) [2 分] 在 Figure-1 上令源点 $s=v_1$，求 $\mathcal{D}_1\circ d_s(v_2)$。

(b) [2 分] 证明 $\mathcal{D}_k\circ d_s(v_k)=d_s(v_k)$；

(c) [0 分 optional] 证明 Dijkstra 算法的正确性。提示：1. 对每一个结点 u 采用函数 $d(u)$ 来估计 $d_s(u)$ 的上界，即恒有 $d_s(u)\le d(u)$；2. 尝试证明 $\mathcal{D}_k\circ d(v_k)=d_s(v_k)=min_{v\in V-\{v_0,...,v_{k-1}\}}\{d_s(v)\}\le min_{v\in V-\{v_0,...,v_{k-1}\}}\{d(v)\}$（从而可以导出：执行算子 n 轮后，所有节点到源点的最短距离都能被发掘）。

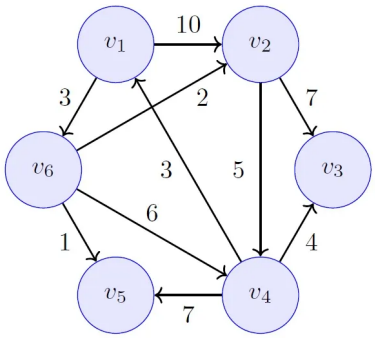

Figure 1: 一个正权有向图 G_1

(d) [2 分] 简单解释 $\mathcal{D}_k$ 的含义。并用 1-2 两句话说明这个算符是怎样在最短路径问题中缩小搜索空间的，其本质思想是什么？

(e) [2 分] 证明题中所给的引理。（本质上这个引理的来源就是动态规划中最优解的性质，后续强化学习中的优化问题也会引用到类似思想）

2.0.2 A^* 算法

引言 相比于必定能找到最短路的 Dijkstra 算法，A* 算法牺牲了路径最短性质的保证，但是从实证意义上提高了两点间某一路径的寻找效率。具体而言，它仅仅在 Dijkstra 的节点上界估计 $d(u)$ 上做了改进，并使用一个新的启发式估计 $d(u) + h(u)$ 来衡量 Closed Set 之外每一个结点的选择价值。其算法可以被陈述如下：

Algorithm 1 A* Path Exploration

Require: A heuristic function $h(u)$ with domain $\mathcal{U}$, source $s \in \mathcal{U}$, terminal $t \in \mathcal{U}$.
Ensure: A sequence $\{P_{st}^{(n)}\} \leftarrow S$ as the path explored.

```
Initialize S ← {s}, d(s) ← 0 and d(u) ← +∞ (∀u ≠ s)
while t ∉ S do
    Set q ← S[−1]
    Update estimations d(u) ← d(q) + ω_qu, ∀u ∈ U − S
    Select u_m ← argmin_{u∈U−S}{d(u) + h(u)}
    Update closed set S ← {S, u_m}
end while
Return S
```

判断下列说法是否正确。如正确，请给出解释或证明；否则请举出反例加以说明。

(a) [2 分] 当 $h(u) := 0$ 时，A^* 算法与 Dijkstra 算法是等价的；

(b) [2 分] 若两个不同的启发式函数 $h_1(u)$ 和 $h_2(u)$ 被分别传入两个算法，则两个算法执行过程中的 $d(u)$ 可能是不同的；

(c) [2 分] 由于 d 是 h 的函数，因而存在某个启发式估计 h_0 和（对于这样的 h_0 而言的）某个结点 $u \in \mathcal{U}$，使得 $d(u)$ 在算法的某一时刻不为图中任一路径的长度。

(d) [2 分] 若图结构的顶点数为 n，边数为 m，则上述定义的 A^* 算法存在时间复杂度为 $O(m + nlogn)$ 的代码实现。

(e) [2 分] 若启发式估计 h 满足：只要对于 $\forall u, v \in \mathcal{U}$ 使得 $d_s(u) + h(u) \leq d_s(v) + h(v)$，就有 $d_s(u) \leq d_s(v)$。那么 A^* 算法和 Dijkstra 算法是等价的；

2.1 网格城市（10 分）

考虑一个由位置坐标 (x, y) 组成的无限城市，其中 x, y 是整数。从每个位置 (x, y) 可以向东、向西、向北或向南走。假如你从 $(0, 0)$ 出发，并想去 (m, n)。我们可以定义以下搜索问题：

- $s_{\text{start}} = (0, 0)$
- $\text{Actions}(s) = \{(+1, 0), (-1, 0), (0, +1), (0, -1)\}$
- $\text{Succ}(s, a) = s + a$
- $\text{Cost}((x, y), a) = 1 + \max(x, 0)$（$x$ 越大成本越贵）
- $\text{IsEnd}(s) = 1[s = (m, n)]$

(a) [4 分] 最低成本路径是什么？它是否唯一（即是否有多条路径可以实现最低成本）？

(b) [6 分] 统一成本搜索（UCS）将如何处理此问题？判断下列说法对 (T) 错 (F)；如果叙述错误请简要说明原因。

- UCS 永远不会终止，因为状态的数量是无限的。
- UCS 将返回最低成本路径，并仅探索 $\{(x, y) | x \in [0, m], y \in [0, n]\}$
- UCS 将返回最低成本路径，并仅探索过去成本低于最低成本的位置

2.2 查找最短路径（12 分）

我们首先从寻找从起点位置（start location）到某个终点位置的最短路径的问题开始。在百度地图中，您只能指定特定的终点位置（例如图书馆）。在这个问题中，我们希望通过指定一组“标签 (tags)”，从而给用户指定多个可能的终点位置的灵活性。（例如，你可以说你想去任何能运动的地方，而不是像东区操场这样的特定位置）。

(a) [8 分] 实现 `ShortestPathProblem`,从而给定一个 `startLocation` 和 `endTag`,最小代价路径对应从 `startLocation` 到任何有 `endTag` 的位置的最短路径。你需要实现 `startState()`,`isEnd(state)`,`successorsAndCosts(state)`。回想一下搜索问题（建模）和搜索算法（推断）之间的是可以分离开来的。你应该专注于建模（定义 `ShortestPathProblem`）；默认的搜索算法 UniformCostSearch (UCS) 在 `util.py` 中实现。

(b) [4 分] 运行 `python mapUtil.py > readableStanfordMap`，将斯坦福地图上可能的位置及其标记写入一个 (较长的) 文件。每个标签都是一个 [key]=[value]。下面是一些关键字的例子：

- `landmark`：手工定义的地标（来自 `data/stanford-landmarks.json`）
- `amenity`：各种便利设施类型（例如，“公园”，“食物”）
- `parking`：各种停车选项（例如，“地下停车场”）

选择一个起始位置和结束标记（可能与您的日常生活相关）并填写 `getStanfordShortestPathProblem()`，以创建搜索问题。然后运行 `python grader.py 1b-custom` 以生成 `path.json`。生成后，运行 `python visualization.py --path-file path.json` 以可视化（在浏览器中打开）。尝试不同的开始位置和结束标记。选择与以下内容对应的两个设置：

- 一个开始位置和结束标签，为在校园里旅行提供了新的灵感（我理解的是尽量长且能穿过大半个校园的路径?）描述这个系统是否对你有帮助。
- 一个开始位置和结束标签，使得系统提供的路径一点也不好玩（我理解的是非常短?）这是由于不正确的建模假设吗？请加以解释。

2.3 查找带无序途径点的最短路径（20 分）

让我们接触一个更强大的功能：无序途径点 (unordered waypoints)！在谷歌地图中，您可以指定一个路径必须经过的序列。例如，从 A 点到 X 点再到 Y 点到 B 点，其中 X,Y 是“途径点”（例如食堂或教学楼）。但是，我们要考虑途径点无序的情况：A → X → Y → B 和 A → Y → X → B 都是允许的。此外，X、Y 和 B 分别由一个标签 (tag) 指定，如问题 1 中所示（例如 ‘amenity=food‘）。如果您考虑自己的日常生活，就会发现这是一个很有用的功能：在西区上了一上午课后，您可能正在回家的路上，但是需要去菜鸟驿站拿包裹，去肯德基疯狂星期四，去书店去买一些笔记本。获得一条短而快速的路径以途径所有这些地方会非常方便（而不是靠自己搜索安排路径）。

(a) [12 分] 实现 `WaypointsShortestPathProblem`，以便给定一个 `startLocation`，一组 `waypointTags` 和一个 `endTag`，最小代价路径对应于从 `startLocation` 到具有 `endTag` 的位置的最短路径，这样所有的 `waypointTags` 都被路径中的某个位置覆盖。注意，单个位置可以用于满足多个标记。和问题 1 一样，你需要实现 `startState()`、`isEnd(state)` 和 `successorsAndCosts(state)`。有很多方法可以实现这个搜索问题，所以你应该仔细考虑如何设计你的状态。我们想要优化一个紧凑的状态空间，使搜索尽可能高效。

(b) [4 分] 如果有 n 个地点和 k 个途径点标签，UCS 可以访问的最大状态数是多少？（请用含 n 和 k 的数学表达式回答，并附上相应的解释来说明合理性）

(c) [4 分] 选择一个起始位置、一组途径点标签和一个结束标签，并编写 `getStanfordWaypointsShortestPathProblem()` 以创建一个搜索问题。与问题 1b 类似，运行 `python grader.py 2c-custom` 以生成 `path.json`,然后进行可视化它并在浏览器中打开。本题中 ** 请选用与问题 1b 相同的两组 `startLocation` 和 `endTag`，并将截图与 1b 对比 **，说明它们的区别是什么，这个新功能是否使路线更加有趣。

2.4 使用 A* 加快搜索速度（30 分）

在这个问题中，我们将探索如何通过使用各种 A* 启发式算法减少需要扩展的状态数量来加快搜索速度。

(a) [10 分] 在此问题中，你将实现 A* 以加快搜索速度。回想一下在课堂上我们提到，A* 只是具有不同成本函数（一个新的搜索问题）的 UCS，因此我们将通过转化成 UCS 来实现 A* 算法。你需要实现 `aStarReduction()`，它将一个搜索问题和一个启发式函数作为输入，并返回一个新的搜索问题。

(b) [10 分] 现在我们可以通过简单地实现各种启发式算法来加快搜索速度。首先，为问题 1 实现 `StraightLineHeuristic`，返回状态与包含任何 tag 的位置的直线距离（the straight line distance from a state to any location with the end tag）。注意，您可能得预先计算一些东西，以便快速计算启发式函数。

(c) [10 分] 接下来，为问题 2 实现 `NoWaypointsHeuristic`，它返回从一个状态到具有结束标记的任何位置的最小成本路径，但忽略所有途径点（本质上是问题 1 的解决方案，因此如果您愿意，可以重复使用它）。注意：您可能希望预先计算一些东西，以便快速评估启发式函数。

3. 体验反馈 [10%]

引言 你可以写下任何反馈，包括但不限于以下几个方面：课堂、作业难度和工作量、助教工作等等。

必填 你在本次作业花费的时间大概是？

选填 你可以随心吐槽课程不足的方面，或者给出合理的建议。若没有想法，直接忽略本小题即可。

附：一些可能有用的参考

- cs221: route，本次实验改编自 CS221 的 route 实验
- Introduction to the A* Algorithm，各种搜索算法的可视化，可交互；
- Learn LaTeXin 30 minutes，简单的 LaTeX 入门